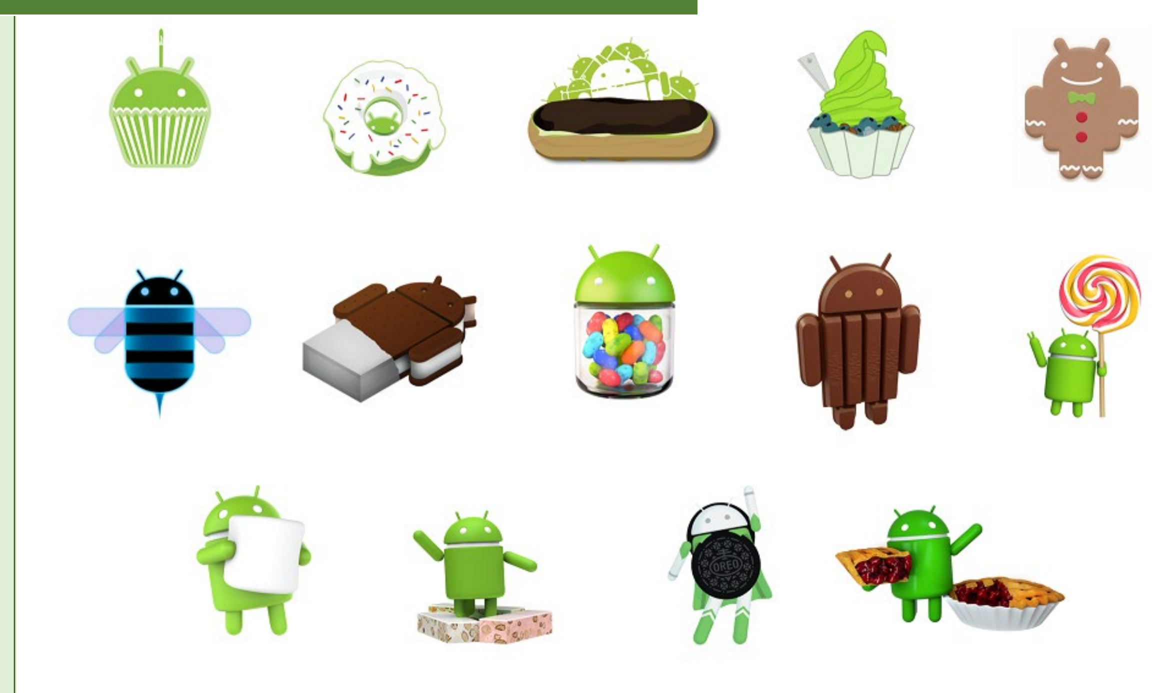


Revista
CelerSMS
Artículos Técnicos
Año 1 / N° 1 / 2021/05/30
BOGOTÁ
Android, El SO abierto para Desarrolladores
ISSN 2745-2336
9 772745 233609

CONTENIDO

Android: lanzamiento en 2008



"No es la especie más fuerte la que sobrevive, ni la más inteligente, sino la que responde mejor al cambio"

Esta frase se aplica perfectamente a Android, el Sistema Operativo más usado en dispositivos móviles en el mundo. Por cierto, esta frase se atribuye comúnmente a *Charles Darwin*, aunque quién la pronunció por primera vez fue el profesor *Leon C. Megginson* en 1963.[A]

EDITORIAL: Historia y Actualidad de Android en Colombia

ARTÍCULOS

DIRECTORIO

La revista CelerSMS es una publicación temática semestral acerca de las tecnologías de telecomunicaciones, plataformas móviles y programación de sistemas. Todo el material publicado en CelerSMS está disponible de manera gratuita y no incluye información comercial o publicitaria. La revista puede ser descargada de la dirección web oficial:

www.celersms.com/revista/

Estos contenidos también pueden ser consultados por medio de Google News:

news.google.com/publications/CAAqBwgKMMKklQswu4WrAw



Las opiniones expresadas por los autores no necesariamente reflejan la postura del editor de la publicación.

Información de contacto:

info@celersms.com
(+57)3023436167
BOGOTÁ

ISSN 2745-2336
ISNI 0000 0005 0265 593X



EDITORIAL

El primer teléfono Android llegó a Colombia en 2010. En ese entonces muchos se preguntaban por qué Google decidió competir con su propio sistema operativo móvil en un mercado dominado por *Symbian*, *BlackberryOS*, *Windows Mobile* (*Windows Phone* aún se llamaba *Windows Mobile*), entre otros sistemas operativos que hoy en día están prácticamente extintos. El gran éxito de Android se debió a su posicionamiento como un sistema operativo abierto. Los desarrolladores podían consultar el código fuente de las distintas capas e interfaces del sistema operativo. Además, las restricciones de seguridad eran mínimas.

Es importante aclarar que hoy en día Google aplica controles más estrictos, tanto a nivel de Android como en los procesos de validación en la tienda en línea Google Play, para mitigar las vulnerabilidades o usos potencialmente no deseados que podrían afectar a los usuarios. Por ejemplo, actualmente resulta muy difícil conseguir autorización de Google para publicar una aplicación que realice llamadas de voz o envío de mensajes SMS, aun con el pleno consentimiento del usuario. En cambio, hasta hace 2 años esto era posible.

En la mayor parte del mercado de Latinoamérica, incluyendo a Colombia, Android es el sistema operativo más utilizado en los teléfonos móviles. Según el reporte *Digital 2021*[B], aproximadamente 9 de cada 10 colombianos con internet móvil utilizan un dispositivo Android.

En 2015 se reportó aproximadamente 1.300 marcas y más de 24.000[C] modelos de *smartphone* y *tablets* Android. Además, Android es utilizado en las consolas de televisión (ej: AndroidTV, FireTV), sistemas de entretenimiento, dispositivos vestibles o *wearables* (ej: pulseras, relojes, gafas inteligentes), realidad aumentada, entre otros.



Si revisamos la lista de las 200[D] apps Android más descargadas a nivel mundial en la tienda de Google Play, podemos notar que no se incluye ninguna app colombiana. Sin embargo, en 2020 Colombia mostró el mayor crecimiento en número de descargas móviles en la región.[E] Entonces, el consumo de apps en Colombia crece más rápido que la oferta de desarrollo nacional. Además, es evidente que el mercado de apps es altamente competitivo.

En esta primera edición de CelerSMS buscamos compartir artículos de desarrollo para Android, cubriendo temas no convencionales, como la compilación sin entorno integrado (IDE), técnicas para mitigar la ingeniería inversa. Esperamos que estos contenidos sirvan para inspirar y orientar a la comunidad de desarrolladores Android hispano-parlantes.

Dedico esta primera edición al Profesor Efraín Barbosa Rojas, científico extraordinario colombiano. Sus sabias palabras marcaron la historia de varias generaciones de físicos, ingenieros, programadores.

Vladimir Kameñar
Editor



Mitigación de Ingeniería Inversa en Apps Android

Victor Celer

No es posible evitar que una aplicación móvil sea descompilada y su funcionamiento interno sea expuesto. El propósito de la ingeniería inversa puede ser el uso ilegal del aplicativo (piratería informática), explotación de sus posibles vulnerabilidades, entre otros efectos perjudiciales para el desarrollador y los usuarios legales. Existen métodos de protección, los cuales encriptan el código del aplicativo, pero dicho código eventualmente es desencriptado por el mismo aplicativo para poder ser ejecutado en la máquina virtual *Dalvik* de Android. Por eso, el título de este artículo hace referencia a la mitigación, no a la prevención. Los desarrolladores de apps Android, especialmente las apps comerciales, pueden tomar medidas para que la ingeniería inversa resulte ser más compleja y dispendiosa. De esta manera se logra diferir y minimizar el impacto. Por ejemplo, si fuera necesario invertir varias semanas de esfuerzo para romper la protección anti-copia de un videojuego, es probable que una nueva versión del juego sea liberada antes de que se logre piratear la anterior.

No confíe en las soluciones genéricas

Un error común consiste en asumir que una protección comercial de tipo DRM o afín puede ser más afectiva que desarrollar un sistema anti-copia propio. El uso de librerías comerciales para protección de apps puede simplificar la labor de los piratas. Tan pronto se identifique la referencia de la librería comercial, una simple búsqueda en los foros especializados en ingeniería inversa podrá arrojar los pasos exactos para desactivar dicha protección. En cambio, crear un sistema anti-copia propio puede complicar mucho más este proceso. Algunas compañías de software han contratado a hacker éticos para crear sus propios sistemas de protección avanzados. Por ejemplo, en los años 2000 la compañía canadiense SSG había contratado los servicios del hacker ético Kris Kaspersky[F] para implementar un sistema anti-copia de CD.

Inclusive sin recurrir al conocimiento de los hacker más expertos, es posible implementar mecanismos anti-piratería efectivos. A continuación se comparten recomendaciones que han demostrado su efectividad en diversos proyectos de apps.

La estrategia del señuelo

Una buena táctica es hacerle creer al *reverser*[1] que el algoritmo es simple. Por ejemplo:

```
if(!isRegistered()){
   Toast.makeText(ctx,
      "Error de registro", 0).show();
   // ...
}
```

[1] En inglés, *reverser* hace referencia a quien practica la ingeniería inversa

Una manipulación mínima de este código sería suficiente para que la copia pirata del aplicativo funcione como si estuviera debidamente registrada. Eso es lo que debería de creer quién haya descompilado este código. Lo ideal es que el aplicativo realmente funcione de manera correcta durante algún tiempo. Por ejemplo, la compañía de videojuegos Aterdux había sacado al mercado un juego con esta misma táctica del señuelo. Los piratas lograron *hackearlo* rápidamente, pero luego se supo que la versión pirata funcionaba sólo hasta cierto punto de la campaña del juego. Más adelante se realizaba una nueva comprobación de autenticidad. Entonces, lo que realmente lograron los piratas fue convertir el juego en una especie de "demo".

La estrategia del señuelo, si se usa de manera inteligente, puede generar una gran frustración en quienes pretenden usar la app de manera ilegal.

Trasladar la funcionalidad de la app

Esto significa alojar de manera externa parte o toda la funcionalidad del aplicativo. El concepto no es nuevo. Por ejemplo, en los años 90 la compañía Rainbow Tech comenzó a alojar bloques funcionales de aplicativos ejecutables en dispositivos llamados *dongle*,[G] los cuales se conectaban al puerto de impresora del PC. La misma idea se aplica hoy en día en los dispositivos de seguridad llamados *token*. En el caso de un teléfono móvil la SIMcard puede hacer las veces de *token*. Es muy difícil replicar la información que se encuentra almacenada en el interior de un dispositivo de seguridad. Algunos proveedores como Ubirch implementan algoritmos de blockchain utilizando la SIMcard. Esta tecnología se desarrolló para garantizar la autenticidad (ej: el registro de un usuario en la red celular, la propiedad de una cuenta bancaria, la telemetría, etc.) El mismo principio se puede usar para proteger la licencia de una aplicación.

Sin embargo, puede ser más simple y efectivo trasladar una parte o toda la funcionalidad de la app hacia la nube. La app podría ser una simple interfaz de usuario para acceder a un servicio que se ejecuta en un servidor en Internet.

Esta estrategia es utilizada por las compañías de desarrollo de software más grandes. Además, se puede decir que las apps de los bancos funcionan de ese modo, ya que las transacciones son procesadas por los sistemas bancarios en línea.

No en todos los casos puede ser posible que la app tenga acceso a Internet para verificación de autenticidad. Por ejemplo, el acceso a Internet puede ser limitado o demasiado costoso a bordo de un avión, entre otros medios de transporte. Otro detalle importante a tener en cuenta es que una falla en la nube puede dejar sin servicio a todos los usuarios legales del aplicativo.

Minimizar el impacto



Si la app se actualiza frecuentemente y su costo es bajo, esta app no sería particularmente atractiva para los piratas. Sin embargo, inclusive las apps gratuitas pueden ser distribuidas ilegalmente. Los piratas pueden contaminar las apps con diferentes tipos de *malware* para luego sustraer la información personal de los usuarios, claves, etc.

Se recomienda generar conciencia en los usuarios, explicando los riesgos que conlleva la instalación de apps provenientes de fuentes no oficiales.



¿Cómo Crear un APK para Android desde Línea de Comandos?

Vladimir Kameñar

Vamos a crear un APK "*Hola Mundo*" para Android utilizando únicamente herramientas de línea de comandos, sin Entorno Integrado de Desarrollo (IDE). Para que este proceso sea lo más simple posible, no usaremos *Gradle*, *Proguard* ni *Bundle Tool*. Omitir estas herramientas no es una buena práctica, pero es aceptable para nuestro APK minimalista. Las únicas herramientas necesarias serán el Kit de Desarrollo Java (JDK) y el SDK de Android. En el presente artículo usaremos la línea de comandos de Windows, pero el mismo proceso se puede adaptar a cualquier otro entorno.

¿Por qué no usar IDE?

Un Entorno Integrado de Desarrollo (IDE) como *Android Studio* permite automatizar todos estos pasos en un solo clic, resaltar la sintaxis del lenguaje, asistir en la depuración, entre muchas otras funciones útiles. Ignorar estos beneficios y recurrir a la línea de comandos no es algo recomendable, pero existen algunas razones que lo pueden justificar:

- El desarrollo de un IDE propio u otro tipo de herramienta que genera APK.
- La necesidad de tener mayor control sobre el proceso de generación de los binarios para evitar al máximo el uso de código autogenerado.
- Los desarrolladores "*old school*" rechazan el uso de entornos gráficos.

Otra razón podría ser la curiosidad por conocer internamente el proceso de generación de APK.

Las herramientas de trabajo

Como se mencionó más arriba, vamos a utilizar únicamente JDK y Android SDK. JDK es necesario para compilar el código Java y empaquetar los binarios. También se puede usar para generar el certificado necesario para firmar la versión productiva (conocida como "*release*") del APK. El SDK de Android incluye las herramientas necesarias para convertir el *bytecode* al formato Dex usado por la máquina virtual Dalvik de Android, alinear y firmar el APK.

Si aún no tiene JDK 1.6 o posterior instalado, lo puede descargar desde el sitio oficial de Oracle.[2] Asegúrese de instalar el Kit de Desarrollo Java (JDK), no solamente el entorno de ejecución (JRE o *Java SE Runtime*). Una buena práctica consiste en configurar la variable de entorno `JAVA_HOME` apuntando a la dirección de instalación de Java (sin incluir el subdirectorio *bin*). Esto no es mandatorio, pero ayuda a evitar conflictos en la localización del JDK, especialmente si existen más de una versión de JDK instaladas.

[2] www.oracle.com/java/technologies/javase-downloads.html

La instalación del SDK de Android no es una tarea tan sencilla. Google ya no provee el paquete de instalación completo para el SDK. Existen 2 opciones alternas:

- Instalar Android Studio desde el sitio oficial.[3] Desde la página inicial del IDE se puede realizar la instalación del SDK fácilmente.
- Otra opción consiste en descargar e instalar las herramientas de línea de comandos (*Command Line Tools*), las cuales están disponibles en la sección más inferior de la misma página oficial.**¡Error! Marcador no definido.** Las herramientas de línea de comandos no son el SDK. Luego de instalar estas herramientas de línea de comandos puede ejecutar el *sdkmanager*[4] para descargar y configurar el SDK de Android.

Ejemplo de proyecto



Un ejemplo de proyecto completo, incluyendo el código fuente Java y los archivos de procesamiento por lotes de Windows para crear y ejecutar el APK, está disponible en nuestro repositorio de GitHub.[5]

Se recomienda crear su propio certificado para firmar el APK. La herramienta *keytool* de JDK se puede usar para reemplazar nuestro certificado de ejemplo con uno propio:

```
del src\demo.keystore /q

REM El siguiente comando es una sola linea
"%JAVA_HOME%"\bin\keytool -genkey -keystore src\demo.keystore -keyalg RSA -keysize 2048 -
validity 10000 -alias demo
```

La ruta *src\demo.keystore* se puede cambiar si desea almacenar el certificado en una ubicación distinta. Si no ha configurado la variable de entorno `JAVA_HOME`, se debe especificar la ruta completa al ejecutar *keytool*. El tiempo de validez del certificado y el nombre del alias se pueden modificar. Keytool requiere ingresar unos datos acerca de la organización, ubicación y solicitará asignar una contraseña. La contraseña asignada en nuestro *demo.keystore* de ejemplo es *password* (se recomienda elegir una combinación de caracteres más segura cuando cree su certificado).

build.bat

Antes de ejecutar el archivo de procesamiento por lotes se debe configurar los siguientes parámetros. Cualquier editor de texto plano como el Bloc de Notas (*Notepad*) se puede usar para editar *build.bat*:

- **`BUILD_TOOLS`** es la ubicación de las herramientas *Build Tools* dentro del SDK de Android. Utilice el valor de ejemplo como referencia.

[3] developer.android.com/studio
[4] developer.android.com/studio/command-line/sdkmanager
[5] github.com/celersms/BatchAPK

- **`ANDROID_JAR`** es la ruta de *android.jar* para la versión de API requerida. La versión de API actual es 29. Se recomienda utilizar la última versión de API disponible. Esto no afecta la ejecución del APK en versiones de Android anteriores, a menos que la aplicación utilice características específicas de las nuevas versiones.

Revisemos las acciones que realiza el archivo de procesamiento por lotes paso a paso:

1. Preprocesar los recursos. Este comando procesa los recursos de Android (imágenes, xml, etc.) y genera *R.java* con el mapa de los ID numéricos de cada recurso.

```
"%BUILD_TOOLS%\aapt" package -f -m -J src -M src\AndroidManifest.xml -S res -I "%ANDROID_JAR%"
```

2. Compilar el código fuente Java y generar el código binario (bytecode) correspondiente.

```
"%JAVA_HOME%\bin\javac" -d obj -classpath src -bootclasspath "%ANDROID_JAR%" src\com\celer\hello\*.java
```

3. Convertir el *bytecode* al formato Dex utilizado por la máquina virtual Dalvik de Android.

```
"%BUILD_TOOLS%\dx" --dex --output=bin\classes.dex obj
```

4. Generar el contenedor APK. Este contenedor no está alineado ni firmado.

```
"%BUILD_TOOLS%\aapt" package -f -m -F bin\unaligned.apk -M src\AndroidManifest.xml -S res -I
"%ANDROID_JAR%"
```

5. Agregar el *bytecode* al APK (*cd* es necesario para mantener la estructura correcta del APK).

```
cd bin
"%BUILD_TOOLS%\aapt" add unaligned.apk classes.dex
```

6. Alinear el APK.

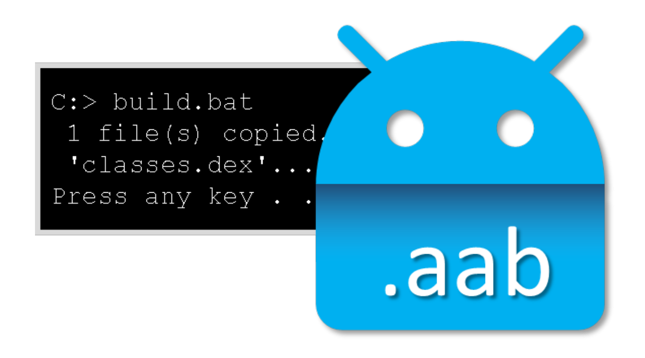
```
"%BUILD_TOOLS%\zipalign" -f 4 unaligned.apk aligned.apk
```

7. Firmar el APK. Se recomienda utilizar tanto el formato de firma V1 como V2 por compatibilidad con todos los dispositivos Android.

```
"%BUILD_TOOLS%\apksigner" sign --ks ..\src\demo.keystore -v1-signing-enabled true -v2-signing-enabled
true --ks-pass pass:password --out hello.apk aligned.apk
```

Si no hay errores el resultado es un APK listo para ser instalado en un dispositivo Android.

En el siguiente artículo aprenderemos a generar paquetes *Android App Bundle* (AAB) desde línea de comandos.



¿Cómo Crear un Paquete AAB para Android desde Línea de Comandos?

Vladimir Kameñar

En el artículo anterior, aprendimos cómo crear un ejecutable APK para Android desde línea de comandos, sin usar ningún IDE. Avancemos más y creemos un paquete AAB (*Android App Bundle*). Sólo utilizaremos JDK, Android SDK y Bundletool. Los pasos que se describen a continuación se probaron en Windows, pero los mismos comandos se pueden adaptar a cualquier otro entorno.

¿Qué es AAB? (*Android App Bundle*)

Actualmente, AAB es el formato preferido para publicar aplicaciones en Google Play. Este formato de publicación incluye todos los recursos y el código compilado de la aplicación. La generación y la firma del ejecutable APK se delegan a Google Play. De esta manera, el APK se puede optimizar al incluir sólo el código y los recursos necesarios para un dispositivo específico. Por lo tanto, el tamaño de la descarga se puede reducir significativamente.

Por ejemplo, supongamos que nuestra aplicación es compatible tanto con teléfonos como con Android TV. Para compatibilidad con Android TV, hemos creado una imagen de fondo grande de alta resolución. Si publicamos la aplicación en formato APK, nuestra imagen de fondo se descargará en los teléfonos inteligentes, aunque nunca se usará allí. El formato AAB permite evitar la inclusión de datos innecesarios para acelerar la descarga, ahorrar ancho de banda y espacio de almacenamiento.

En el momento de redactar este artículo, sólo Google Play utiliza el formato AAB. Las demás tiendas de aplicaciones no lo utilizan.

¿Por qué no usar IDE?

Android Studio puede crear un AAB automáticamente con un solo clic. Hacerlo manualmente puede ser útil para entender mejor cómo funciona este proceso. Por ejemplo, si está creando su propia herramienta para generar aplicaciones de Android, esta guía se puede utilizar como referencia. Hay casos especiales en los que puede ser necesario evitar el uso de código generado automáticamente. Entonces, es posible que deba compilar el paquete manualmente.

Las herramientas de trabajo

Sólo necesitaremos el JDK, Android SDK y Bundletool. JDK es necesario para compilar el código Java y generar los binarios. También se puede utilizar para generar el certificado para la firma. El SDK de Android contiene las herramientas necesarias para convertir el *bytecode* al formato Dex usado por la máquina virtual Dalvik de Android, alinear y firmar el paquete. Bundletool se utiliza

para generar el AAB. También se usa para probar el AAB localmente, generando los APK específicos para cada dispositivo de prueba.

Si aún no tiene JDK 1.6 o posterior instalado, lo puede descargar desde el sitio oficial de Oracle.[6] Asegúrese de instalar el Kit de Desarrollo Java (JDK), no solamente el entorno de ejecución JRE. Se recomienda configurar la variable de entorno `JAVA_HOME` para que apunte a la ubicación de instalación de Java (sin incluir el subdirectorio *bin*). Esto no es obligatorio, pero ayuda a evitar conflictos en la localización del JDK, especialmente si existe más de una versión de JDK instalada.

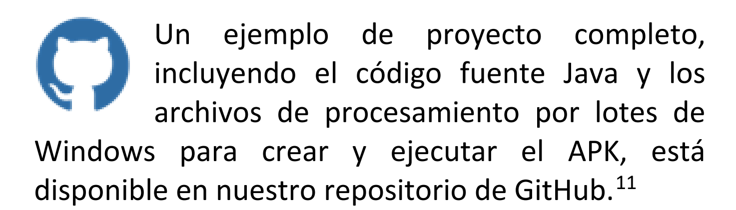
La instalación del SDK de Android no es una tarea tan sencilla. Google ya no provee el paquete de instalación completo para el SDK. Existen 2 opciones alternas:

- Instalar Android Studio desde el sitio oficial.[7] Desde la página inicial del IDE se puede realizar la instalación del SDK fácilmente.
- Otra opción consiste en descargar e instalar las herramientas de línea de comandos (*Command Line Tools*), las cuales están disponibles en la sección más inferior de la misma página oficial.**¡Error! Marcador no definido.** Las herramientas de línea de comandos no son el SDK. Luego de instalar estas herramientas de línea de comandos puede ejecutar el *sdkmanager*[8] para descargar y configurar el SDK de Android.

Las herramientas *Build Tools* del SDK de Android deben tener al menos la versión 29.0.2. Es posible que las versiones anteriores no incluyan la herramienta AAPT2 o que existan problemas de compatibilidad. También puede descargar la última versión de AAPT2 de Maven como se explica aquí.[9]

La versión más actual de Bundletool se puede descargar aquí.[10] La herramienta es un único archivo JAR.

Ejemplo de proyecto



Un ejemplo de proyecto completo, incluyendo el código fuente Java y los archivos de procesamiento por lotes de Windows para crear y ejecutar el APK, está disponible en nuestro repositorio de GitHub.[11]

Al igual que en el artículo anterior, se recomienda crear su propio certificado para firmar el paquete. La herramienta *keytool* de JDK se puede usar para reemplazar nuestro certificado de ejemplo con uno propio, como se explica en el artículo anterior.

build.bat

Antes de ejecutar *build.bat* se debe configurar los parámetros que se describen a continuación. Un editor de texto plano como el Bloc de Notas (*Notepad*) se puede usar para editar *build.bat*:

- **`BUILD_TOOLS`** es la ubicación de las herramientas *Build Tools* dentro del SDK de Android. Utilice el valor de ejemplo como referencia.
- **`ANDROID_JAR`** es la ruta de *android.jar* para la versión de API requerida. Se recomienda utilizar la última versión de API disponible. Esto no afecta la ejecución del APK en versiones de Android anteriores, a menos que la aplicación utilice características específicas de las nuevas versiones.
- **`BUNDLETOOL`** es la ubicación del archivo jar de Bundletool. Intente utilizar la última versión, si es posible.

[6] www.oracle.com/java/technologies/javase-downloads.html
[7] developer.android.com/studio
[8] developer.android.com/studio/command-line/sdkmanager
[9] developer.android.com/studio/command-line/aapt2
[10] github.com/google/bundletool/releases
[11] github.com/celersms/BatchAAB

Revisemos las acciones que realiza el archivo de procesamiento por lotes paso a paso:

1. Compilar los recursos. Este comando analiza los recursos (imágenes, xml, etc.) y los empaqueta en un archivo ZIP. Observe que para construir un AAB estamos usando la herramienta AAPT2 en lugar de AAPT.

```
"%BUILD_TOOLS%\aapt2" compile --dir res\ -o obj\res.zip
```

2. Enlazar los recursos. Esta es una sola línea:

```
"%BUILD_TOOLS%\aapt2" link --proto-format -o obj\linked.zip -I "%ANDROID_JAR%" --manifest
src\AndroidManifest.xml --java src obj\res.zip --auto-add-overlay
```

3. Compilar el código fuente Java y generar el código binario (*bytecode*) correspondiente.

```
"%JAVA_HOME%\bin\javac" -d obj -classpath src -bootclasspath "%ANDROID_JAR%" src\com\celer\hello\*.java
```

4. Convertir el *bytecode* al formato Dex utilizado por la máquina virtual Dalvik de Android.

```
"%BUILD_TOOLS%\dx" --dex --output=bin\classes.dex obj
```

5. Combinar los recursos y el *bytecode* en un solo paquete:

```
cd obj
"%JAVA_HOME%\bin\jar" xf linked.zip resources.pb AndroidManifest.xml res
mkdir manifest dex 2>nul
move AndroidManifest.xml manifest
copy /Y /B ..\bin\classes*.dex dex\ 2>nul
"%JAVA_HOME%\bin\jar" cMf base.zip manifest dex res resources.pb
```

6. Generar el AAB.

```
"%JAVA_HOME%\bin\java" -jar "%BUNDLETOOL%" build-bundle --modules=base.zip --output=..\bin\hello.aab
```

7. Firmar el AAB.

```
"%JAVA_HOME%\bin\jarsigner" -keystore ..\src\demo.keystore -storepass password ..\bin\hello.aab demo
```

Si no hay errores el resultado es un AAB listo para ser publicado en Google Play.



Generación de Identificadores Únicos de Dispositivo Android

Victor Celer

Generar u obtener un único identificador para cada dispositivo donde se instala una app puede ser útil para diferentes propósitos. Por ejemplo, en un sistema de votación en línea puede ser interesante evitar múltiples votos en un mismo dispositivo. En los sistemas transaccionales los identificadores únicos sirven para propósitos de trazabilidad. Esto también puede servir para efectos de control de licencias (ej: limitar el número de dispositivos donde se puede instalar la app).

La información personal del usuario, como su número de documento de identidad o número de teléfono, no debería de ser usada para estos propósitos debido a restricciones de confidencialidad. Tampoco se recomienda el uso de identificadores de hardware, como la dirección MAC o el número IMEI.[H] Una de las razones para evitar el uso de identificadores de hardware en Android es que su obtención requiere permisos elevados. Además, el número IMEI no siempre es único. Algunos de estos números se repiten a pesar de que las normas de GSMA lo prohíben.[I]

android_id

No existe un método que arroje un identificador realmente único en todas las versiones de Android. En este artículo combinamos varios de los métodos existentes para cubrir la mayoría de los casos de uso. Podemos comenzar con el método más usado que es el “android_id”:

```
long uid = 0;
String ss;
try{
   ss = Secure.getString(ctx, "android_id");
   if(ss != null){
      uid = new BigInteger(ss, 16).longValue();
      if(uid == 0x9774D56D682E549CL)
         uid = 0; // filtrar valor no único
   }
}catch(Exception ex){
   Log.e("UID", "error", ex);
}
```

Nótese que este código continúa la ejecución en caso de cualquier error inesperado, incluyendo errores numéricos. Si se obtiene el valor hexadecimal **9774D56D682E549C**, este valor es descartado, ya que se conoce que el mismo valor puede ser reportado en múltiples dispositivos.[J]

MAC

Si el anterior código no nos arroja un valor distinto de cero, entonces podemos intentar obtener la dirección MAC de Bluetooth, suponiendo que la app tiene permisos para usar Bluetooth:

```
// Desde Android 10 la MAC no puede ser
// consultada de esta manera
if(uid == 0 && Build.VERSION.SDK_INT <= 28){
   ss = null;
   try{
      BluetoothAdapter btAdapter =
        BluetoothAdapter.getDefaultAdapter();
      try{
         Field ff =
      btAdapter.getClass().getDeclaredField(
         "mService");
         ff.setAccessible(true);
         Object oo = ff.get(btAdapter);
         ss =
   (String)oo.getClass().getDeclaredMethod(
      "getAddress").invoke(oo);
      }catch(Throwable th){
         ss = btAdapter.getAddress();
      }
   }catch(Exception ex){
      Log.e("UID", "error", ex);
   }
   if(ss != null){

      // Convertir MAC a long
      int bb, vv = 0, xx = 0, yy = ss.length();
      boolean pp = false;
      while(xx < yy){
         if((bb = (ss.charAt(xx++) |
               0x20) - 0x30) > 9)
            bb -= 0x27;
         if(bb >= 0 && bb <= 0xF){
            if(pp = !pp)
               vv = bb << 4;
            else
               uid = uid << 8 | vv | bb;
         }
      }

      // MAC no válida
      if((int)uid == 0 || uid <= 0 ||
            uid > 0xFFFFFFFFFFFFL)
         uid = 0;
   }
}
```

Estampa de tiempo (*timestamp*)

Si el código anterior tampoco logra obtener un buen valor, entonces una alternativa puede ser tomar la hora exacta con milisegundos, en la cual se instaló un paquete conocido, como "*SystemUI*". Este valor no es único, pero puede tener una dispersión y persistencia razonablemente buenas:

```
if(uid == 0){
   try{
      uid = new File(
ctx.getPackageManager().getApplicationInfo(
         "com.android.systemui", 0).sourceDir
      ).lastModified();
   }catch(Exception ex){
      Log.e("UID", "error", ex);
   }
}
```

Una combinación de estos 3 métodos permite cubrir casi la totalidad de los casos de uso. Para evitar colisiones entre los 3 o más métodos se recomienda acompañar el identificador único con algún indicador que sirva para diferenciar el método utilizado.

Recuerde que no todos los teléfonos tienen Bluetooth. No todos los dispositivos Android son teléfonos y tienen suscripción móvil e IMEI. No todos los dispositivos Android tienen las mismas apps por defecto disponibles. La diversidad de dispositivos Android genera retos de desarrollo interesantes. Si omitimos compatibilidad con algún dispositivo o grupo de dispositivos podemos perder un segmento del mercado.

Referencias

[A] quoteinvestigator.com/2014/05/04/adapt/ , Quote Investigator (*en inglés*), 2014
[B] wearesocial.com/digital-2021, We are Social y Hootsuite (*en inglés*), 2021
[C] www.opensignal.com/sites/opensignal-com/files/data/reports/global/data-2015-08/2015_08_fragmentation_report.pdf, OpenSignal (*en inglés*), 2015
[D] play.google.com/store/apps/top, Google Play
[E] www.larepublica.co/internet-economy/la-instalacion-de-aplicaciones-en-dispositivos-moviles-ha-crecido-15-en-colombia-3029694, Diario La República, 2020
[F] en.wikipedia.org/wiki/Kris_Kaspersky, Wikipedia (*en inglés*)
[G] es.wikipedia.org/wiki/Dongle, Wikipedia
[H] developer.android.com/training/articles/user-data-ids, Recomendaciones para identificadores únicos
[I] www.gsma.com/latinamerica/wp-content/uploads/2012/05/IMEI-Database-Overview.pdf, IMEI Database (*en inglés*), GSMA
[J] code.google.com/archive/p/odinmobile/wikis/FrequentlyAskedQuestions.wiki



"Hecho es mejor que perfecto"
Mark Zuckerberg